Episode 4.08 – DeMorgan's Theorem

Welcome to the Geek Author series on Computer Organization and Design Fundamentals. I'm David Tarnoff, and in this series, we are working our way through the topics of Computer Organization, Computer Architecture, Digital Design, and Embedded System Design. If you're interested in the inner workings of a computer, then you're in the right place. The only background you'll need for this series is an understanding of integer math, and if possible, a little experience with a programming language such as Java. And one more thing. Our topics involve a bit of figuring, so it might help to keep a pencil and paper handy.

Some of you may have noticed that the truth table for the AND gate and the truth table for the OR gate have a similar characteristic: regardless of the number of inputs to the gate, each has a single row with a binary output that is different from the binary output of all the remaining rows. The AND operation has zeros in all rows except the one row where all of the inputs equal one. The OR operation has ones in all rows except the one row where all of the inputs equal zero. Okay, so maybe they're not exactly the same, but consider the possibility that we could manipulate one of the tables to match the other by incorporating some inverters into the circuit.

Let's start by placing inverters at each of the inputs going into the AND gate. The collective outputs from these inverters changes the top row of the truth table from all zeros to all ones, and the values decrement to all zeros at the bottom row. In essence, what we've done is reverse the order of rows. Picture this for the truth table for a three input AND gate. The operation changes to $A \cdot B \cdot C$ (read A-bar and B-bar and C-bar).

Comparing the truth table for this modified AND expression with the truth table for the three-input OR gate reveals that they are similar, but not yet the same. The two truth tables are now inverses of one another. So let's invert the output of the OR gate to get $\overline{A + B + C}$ (read as the inverse of the sum of A OR B OR C). Now both truth tables are the same. In other words, an AND with inverted inputs equals an OR with an inverted output. A similar truth table-based proof can be used to show that the output of an OR gate with inverted inputs equals the inverted output of an AND gate with non-inverted inputs. This property, where the inverted output of the AND operation equals the output of the OR operation with inverted inputs and where the inverted output of the OR operation equals the output of the AND operation with inverted inputs, is call DeMorgan's Theorem.

In Episode 4.07 – Identities of Boolean Algebra, we discussed how the AND and OR operations are analogous to multiplication and addition in traditional algebra. The inverse, however, does not have an analogous operation. Unlike the unary operation negation, the NOT gate cannot be distributed to the inputs of an AND or an OR gate. That's why we need DeMorgan's Theorem. It allows us to distribute an inverter from the output of an AND or an OR gate across all of the gate's inputs. This distribution flips the operation of the gate – an AND gate is switched to an OR gate and an OR gate is switched to an AND gate.

Since DeMorgan's Theorem can only be distributed across a single operation, either AND or OR, at one time, it can become difficult to handle when there is a mixture of operations below the bar. In Episode 4.05 – Introduction to Boolean Algebra, we discussed the application of the order of operations: parenthesis, then inverse, then AND, and lastly OR. Distributing an inverse to the inputs of a mixture of

operations, however, requires us to maintain the order of operations within the Boolean expression being inverted. This means we need to be careful to apply DeMorgan's Theorem in the reverse order of operations below the bar. Maybe we'll be able to describe this better using an example.

Assume we wish to invert the result of the Boolean expression A + B · C. The expression itself has an order of operations. First, AND B with C, then OR the result of that with A. If we were to draw the circuit, from left to right, the inputs B and C first pass through the AND gate. The output of the AND gate is then combined with A in the OR gate. Now place an inverter at the output of the OR gate.

If we wish to distribute the inverter at the output of this circuit back to the inputs, what is the first gate we need to distribute the inverter through using DeMorgan's Theorem? Well, it's the OR gate. If we "push" that inverter back towards the OR gate passing it through to the inputs, the inverter is distributed to the inputs while the operation changes from OR to AND. Our new circuit looks like this. The inputs B and C first pass through the AND gate, then pass through an inverter going into one of the inputs of the newly created AND gate. The other input to our newly created AND gate is an inverted signal A.

The new circuit's operation first ANDs B with C, then inverts the result, then ANDs it with the inverse of A. The new Boolean expression would be A · (B · C) (read as A-bar AND-ed with the inverse of B AND C). We still have one more gate to pass through before we've distributed the inverters all the way back to the inputs. We need to distribute the bar across the AND of B AND C. When we "push" the inverter back through B · C, inverters are distributed to both B and C, and the AND operation becomes an OR. This makes our final expression A · (B + C) (read as A-bar AND-ed with the sum of B-bar OR C-bar). Be careful of the parenthesis in this expression. B-bar OR C-bar must be reckoned before being AND-ed with A-bar. If we neglect to include the parenthesis, order of operations would suggest we AND A-bar with B-bar before OR-ing with C-bar. Just remember that to guarantee that the inverters are being distributed properly, we must apply DeMorgan's Theorem in the reverse order of operations for the Boolean expression that's included under the bar.

Let's revisit a Boolean expression we presented in Episode 4.05 – Introduction to Boolean Algebra. Toward the end of that episode, we used a truth table to show that for every possible value that A and B can take on, the result of B · (A + B) (read B AND-ed with the inverse of the sum A OR B) is zero. Now that we have a few properties of Boolean algebra in our toolbox, let's show how this expression breaks down.

Begin by observing that the sum, A + B, is being inverted. Using DeMorgan's Theorem, we can distribute the bar across A OR B to get A · B (read A-bar AND B-bar). This gives us the expression B · (A · B) (read B AND-ed with A-bar AND B-bar). Using the Commutative Law, we can swap A-bar and B-bar to get the expression B · (B · A) (read B AND-ed with B-bar AND A-bar). Now that we have three signals, B, B-bar, and A-bar, being combined across the common operation of AND, we can drop the parenthesis using the Associative Law to get B · B · A (read B AND B-bar AND A-bar). From Episode 4.07 – Identities of Boolean Algebra, we have the inverse law for the AND operation which states that any signal AND-ed with its inverse equals zero. That means that B AND B-bar must equal zero. This gives us the expression 0 AND A-bar. The same episode presents the annulment law for the AND operation, which states that any signal AND-ed with zero equals zero. Therefore, 0 AND A-bar equals zero, and we have proved using the properties of Boolean algebra that B · (A + B) (read B AND-ed with the inverse of A OR B) is zero.

In our next episode, we will use some of the methods presented over the past three episodes to do more of this simplification stuff. These simplifications will not change the truth tables, but the modified circuits may perform better either in the area of speed, power consumption, or cost. Later in this series, we will also use these properties of Boolean algebra to create two of the standard formats for Boolean expressions. These formats allow for quicker operation and support the use of programmable hardware components. This also gives us a technique to convert truth tables into circuitry. It will be our first foray into designing circuitry based on a system specification.

For transcripts, links, or other podcast notes, please visit us at intermation.com where you will also find links to our Instagram, Twitter, Facebook, and Pinterest pages. Until the next episode, remember that while the scope of what makes a computer is immense, it's all just ones and zeros.